Array Organization in Lock-Step Parallel Memories

Mayez A. Al-Mouhamed[1], Steven Seiden[2], and SHussam Abu-Haimed[3]

***Abstract*—The serialization of memory accesses is a major limiting factor in high performance SIMD computers. The data patterns or *templates* that are accessed by a program can be perceived by the compiler, and therefore, the design of dynamic storage schemes that minimize conflicts may dramatically improve performance.**

The problem of finding storage schemes that minimize the access time of arbitrary sets of power-of-2 data patterns is proved to be NP-complete. We propose linear address transformations that can be dynamically applied by each processing element for mapping array references onto memories. An efficient approach for *combining* the constraints of different access patterns into *one single linear address transformation* is presented. We prove that finding the transformation that minimizes the access time is reduceable to N-coloring, where N is the number of parallel memories. Using coloring heuristics, storage schemes that minimize overall memory access time are heuristically constructed. Evaluation shows that synthesized storages lead to dramatic reduction of the degree of conflict and largely outperforms interleaving and row-column-diagonals (STARAN) storages.

Keywords: Heuristics, memory organization, parallel memories, performance evaluation, storage schemes

I. Introduction

The serialization of memory accesses is a major limiting factor to bandwidth balancing between processors and parallel memories. Memory interleaving maps consecutive addresses into different physical memories so that simultaneous accesses can be performed in one memory cycle. Conflict-free access is possible only when the stride associated with successive references is relatively prime to the number of memories.

Budnik and Kuck [1] proposed the *row-rotation scheme* that allows conflict-free access to arbitrary row, and column of arrays. To avoid run-time overhead, Sohi [2] proposed bit-wise boolean address transformations for vector processors in order to determine the memory number where a given array element should be stored. The scheme can be efficiently used for power-of-2 strides but other strides can also be accessed through the use of few buffers at the memory inputs and outputs. The buffers reduce the effects of transient degradation.

The image by the storage scheme of all the elements of a given pattern should map into different memories. Therefore, the columns or the rows of the needed transformation matrix should be linearly independent [3]. As the interconnection network should provide data alignment between processors and memories, other constraints can then be used for finding the storage matrix. Boppana [4] proposed a bitwise linear transformation matrix (non-singular) for accessing to the row, column, main-diagonal, and square blocks.

We are concerned with dynamically reconfigurable storage schemes for SIMD models that minimize the overall access time of an arbitrary set of weighted data patterns. The problem is to find how arrays can be stored into parallel memories in order to enforce the elements of a given data pattern be uniformly distributed over the memories. Given a program that requires access to a set of data patterns, our objective is to find a cost-effective storage scheme that minimizes overall memory access time.

This paper is organized as follows. Section 2 presents the background. The template bases are defined in Sections 3. Section 4 and 5 present the *Perfect Storage* that allows combining data patterns and minimizing the implementation cost. Section 6 presents the *Semi-Perfect storage* that allows minimizing overall access conflicts. Section 7 presents our method for synthesizing storage schemes. The evaluation of this work and comparison to other proposals are presented in Section 8. The conclusions to this work are presented in Section 9.

II. Background

Any processor in an SIMD system can access any memory through an interconnection network. Assume there are an equal number $P = 2^p$ of processors and memories. Conflicts occur when i processors ($i > 1$) try to access the same memory during a given cycle which requires i cycles for the memory to serve them. Since all processors run in lock-step, the entire computation is dramatically slowed. It would be desirable to store the data that should be simultaneously accessed into different memories so that parallel access can be achieved.

A *template* is defined as a pattern of array elements whose addresses are related by some relationship such as those shown in Figure 1. A template is a set of data elements that should be accessed in parallel, by all the PEs, during the running of the program. We are interested in templates having power-of-2 elements. The *origin* of a template is the coordinate, within the array, of its upper left-most element. For example the set of all rows (columns) is a template, and each row (column) is a template instance [3].

The memory is assumed to be a single two dimensional array of size $2^d \times 2^d$ such that the element in the ath row and the bth column is denoted by (a, b). The upper left-most element is $(0, 0)$. The sizes of the horizontal and vertical dimensions are both 2^d.

A row position a can also be thought of as a vector, over the finite field Z_2, the integers modulo 2. In Z_2, addition corresponds to logical *exclusive or*, and multiplication corresponds

(1) Department of Computer Engineering, College of Computer Science and Engineering (CCSE) King Fahd University of Petroleum and Minerals (KFUPM), Dhahran 31261, Saudi Arabia. mayez@ccse.kfupm.edu.sa

(2) Info. & Comp. Sc. Dept.,ICS, UCI, Irvine, CA 92717 seiden@ics.uci.edu

(3) Department of Computer Engineering, CCSE, KFUPM, Dhahran 31261, Saudi Arabia. shams@ccse.kfupm.edu.sa

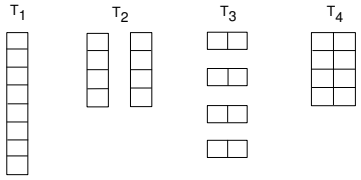

Fig. 1. Example of four templates

to logical *and*. We define a vector space $\mathcal{F} = Z_2^d$ for horizontal position. Let $F = \{f_0, f_1, \dots f_{d-1}\}$ be the canonical basis of $\mathcal{F}$. Each row has a unique representation as a vector in $\mathcal{F}$. A Row a is expressed as $a_0 f_0 \oplus a_1 f_1 \oplus \cdots \oplus a_{d-1} f_{d-1}$ in terms of F. We similarly define vector spaces $\mathcal{G}$ for column positions and $\mathcal{H}$ for memory unit numbers, with canonical bases $G = \{g_0, g_1, \dots g_{d-1}\}$ and $H = \{h_0, h_1, \dots h_{p-1}\}$, respectively.

The Cartesian product of the vector spaces $\mathcal{F}$ and $\mathcal{G}$ is a new vector space $\mathcal{V} = \mathcal{F} \times \mathcal{G}$ with basis $\{f_0, f_1, \dots f_{d-1}, g_0, g_1, \dots g_{d-1}\}$. Let $n = 2d$. We denote this combined basis as $V = \{v_0, v_1 \dots v_{n-1}\}$, where $v_0 = f_0$, $v_1 = f_1$, $v_d = g_0$ etc. This vector space is isomorphic to Z_2^n. Any location (a, b) in memory is uniquely associated with a linear combination of the basis elements $a_0 v_0 \oplus \cdots \oplus a_{d-1} v_{d-1} \oplus b_0 v_d \oplus \cdots \oplus b_{d-1} v_{n-1}$.

III. Templates

We define a template T_i by a basis T_i, which is a non-empty subset of V. We assume all templates bases are of size p, i.e. there are 2^p elements in any given template instance. This is best explained by looking at some examples. Let p and d both be 3. Our basis is $V = \{f_0, f_1, f_2, g_0, g_1, g_2\}$, or alternatively $V = \{v_0, v_1 \dots v_5\}$. Consider the template T_1 defined by $T_1 = \{f_0, f_1, f_2\}$. The set of templates instances described are all non-overlapping columns of eight elements, the upper left-most template instance having origin $(0, 0)$. Every element in a template instance is a linear combination $a_0 f_0 \oplus a_1 f_1 \oplus a_2 f_2 \oplus b_0 g_0 \oplus b_1 g_1 \oplus b_2 g_2$, where the b's are constant, and the a's are allowed to vary ($b_0 g_0 \oplus b_1 g_1 \oplus b_2 g_2$ is the template instance's origin). Intuitively, we are letting the three least significant bits of a vary, while the other bits of a, and all bits of b, remain fixed. By allowing different bits to vary, we generate templates of different shapes. Let T_2 have basis $T_2 = \{f_0, f_1, g_1\}$. Then, in T_2 all template instances are four elements tall. Since g_0 is omitted, this template skips a column. We define T_3 by $T_3 = \{f_1, f_2, g_0\}$. This template is four 1×2 sub-arrays spaced two rows apart. We let T_4 have basis $T_4 = \{f_0, f_1, g_0\}$. All of these templates are illustrated in Figure 1.

An XOR-scheme is a linear function $\phi : \mathcal{F} \times \mathcal{G} \mapsto \mathcal{H}$. The function ϕ is represented by a $p \times n$ matrix, which we denote Φ. We apply ϕ to a vector X by matrix multiplication:

$$\phi(x_0, x_1 \dots x_{n-1}) = \Phi \cdot \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

The ith entry of the jth column of Φ is $\Phi_{i,j}$. (The upper left-most entry is $\Phi_{0,0}$.) We denote the ith column of Φ by $\Phi_{*,i}$, and the ith row of Φ by $\Phi_{i,*}$. The columns of this matrix represent the values, in terms of the basis H, of ϕ on the members of the basis V. I.e. $\Phi_{*,i}$ is the value of $\phi(v_i)$.

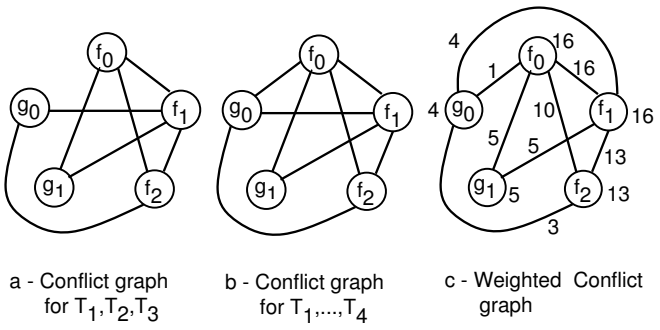

Fig. 2. Conflict graph for template bases

We can also consider ϕ as an ordered set of p functions, $\{\phi_0, \phi_1 \dots \phi_{p-1}\}$, mapping from $\mathcal{F} \times \mathcal{G}$ to Z_2, where $\phi(X) = \phi_0(X) h_0 \oplus \phi_1(X) h_1 \oplus \cdots \oplus \phi_{p-1}(X) h_{p-1}$. The matrix of ϕ_i is $\Phi_{i,*}$.

Then ϕ allows conflict free access to T_i, if and only if ϕ maps each linear combination of T_i to a unique element of $\mathcal{H}$. Since ϕ is linear, all translations of these linear combinations are also conflict free. In other words, if for one template instance X in T_i, ϕ restricted to X is one to one, then for all template instances X in T_i, ϕ restricted to X is one to one.

IV. Perfect XOR-Schemes

We say that an XOR-scheme is *perfect* if and only if all columns $\Phi_{*,i}$ contain at most one non-zero entry. In other words, in the expression $\phi(X) = \phi_0(X) h_0 \oplus \phi_1(X) h_1 \oplus \cdots \oplus \phi_{p-1}(X) h_{p-1}$, any particular x_i is used at most once, where h_i is the ith canonical vector of the basis of the memory numbers. A perfect XOR-scheme only requires n XOR gates to be implemented.

We give an example using the templates of Section III for which $n = 6$ and $p = 3$. We would like to find a perfect XOR-scheme for this set of templates. But first, let us consider the subset of templates $\{T_1, T_2, T_3\}$. Notice that f_1 appears in all templates bases of this subset. Therefore, let $\phi_0(a_0, a_1, a_2, b_0, b_1, b_2) = a_1$. Further, notice that g_0 appears only in T_3 and f_0 appears only in T_1 and T_2. Therefore, let $\phi_1(a_0, a_1, a_2, b_0, b_1, b_2) = b_0 \oplus a_0$. Finally, notice that g_1 appears only in T_2 and f_2 appears only in T_1 and T_3. We let $\phi_2(a_0, a_1, a_2, b_0, b_1, b_2) = b_1 \oplus a_2$. Let us generalize this procedure. Let Φ be the matrix of a perfect XOR-scheme. If $\Phi_{k,i} = 1$ and $\Phi_{k,j} = 1$ where $i \neq j$, and v_i and v_j are both in T_ℓ, then T_ℓ cannot not be accessed conflict free.

Proof: This is easily seen by a counting argument. ϕ_k can only take on two values. x_i and x_j can each take on two values, for a total of four. Further, because each column of Φ contains at most one non-zero, no other ϕ_k varies with x_i or x_j. Therefore, given a template instance, we are mapping two elements of it to each memory unit in its image. Let $T : V \mapsto 2^T$ be a function mapping from a given basis vector to the set of templates bases which contain that vector. More precisely,

$T_i \in T(v)$ if and only if $v \in T_i$. We allow $\Phi_{k,i} = 1$ and $\Phi_{k,j} = 1$ only if $T(v_i)$ and $T(v_j)$ are disjoint. The *conflict graph* (V, E) of the template set represents this relationship. The vertices of the graph are the vectors of the basis V. An edge (v_i, v_j) is in E if and only if $i \neq j$ and $T(v_i) \cap T(v_j) \neq \emptyset$. The graph for T_1, T_2, and T_3 is illustrated in Figure 2-a.

For a set of templates T, a conflict free XOR-scheme for P memory units exists, if and only if the conflict graph of the templates is p-colorable.

Proof of this Theorem can be found in [5]. Indeed, the graph in Figure 2-a is 3-colorable. The matrix of the perfect XOR-scheme is:

$$\Phi = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 & g_0 & g_1 & g_2 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

However, consider what happens when T_4 is added. The graph of $T_1 \ldots T_4$ is shown in Figure 2-b. This graph is not 3-colorable, because g_0, f_0, f_1 and f_2 form a clique, and thus no perfect XOR-scheme exists. We can prove [5] that finding a perfect XOR-scheme for a set of templates and arbitrary p is NP-complete. Note that two-coloring is polynomial, and thus finding a perfect XOR-scheme for 4 memory units is tractable.

V. Non-Perfect XOR-schemes

Suppose we do not restrict ourselves to perfect XOR-schemes. Does the set of templates in Figure 1 have an XOR-scheme? By inspection, we find the matrix of one possible XOR-scheme is:

$$\Phi = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 & g_0 & g_1 & g_2 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Why does this function allow conflict free access? Consider ϕ restricted to the template T_1. The matrix of this restricted function is:

$$\Phi(T_1) = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} \end{array}$$

Notice that this matrix has rank 3; its columns are linearly independent. The dimension of the image of ϕ restricted to T_1 is three, and therefore, conflict free access is assured. We denote ϕ restricted to T_i by $\phi(T_i)$. The matrix of $\phi(T_i)$ is $\Phi(T_i) = (\Phi_{*,j})_{v_j \in T_i}$. The matrices of ϕ restricted to $T_2 \ldots T_4$ have all rank 3. In general, we need to find a function ϕ with matrix Φ, such that $\Phi(T_i)$ has rank $|T_i|$, for all templates T_i. How can such XOR-schemes be found? First consider a more general problem.

Suppose we are given: 1) a vector space $\mathcal{Z} = Z_2^p$, 2) a set of variables $V = \{v_0, v_1, \ldots v_{n-1}\}$ and, 3) a set $T = \{T_1, T_2, \ldots T_t\}$, where each T_i is some member of 2^V and each variable must appear in some T_i. The problem is to assign each variable a value in $\mathcal{Z}$, such that for all i, the vectors assigned to the variables in T_i are linearly independent. We call this problem *linear independence satisfiability* (LIS). We can prove [5] that LIS is NP-complete. Our proof that LIS is NP-complete does not depend on the fact that $\mathcal{Z}$ is over Z_2. LIS is NP-complete over any finite field.

It is easily seen that LIS is equivalent to finding a non-perfect XOR-scheme. Note that finding a general XOR-scheme for 4 memory units is NP-complete, where finding a perfect XOR-scheme for 4 memory units is polynomial. Also note that we can build independence graphs only for $p = 2$. We need to find a model of XOR-schemes for $p > 2$, from which good heuristics can be derived.

VI. Semi-Perfect XOR-Schemes

We investigate a class of XOR-schemes which are somewhere between perfect and general XOR-schemes. We call this class of XOR-schemes *semi-perfect*. We say that an XOR-scheme ϕ, represented by a matrix Φ, is semi-perfect if and only if for all templates T_i in $\mathcal{T}$, the matrix of ϕ restricted to T_i contains at most one column with two non-zero entries, and the rest of the columns have one or zero non-zero entries.

Let Φ be a matrix over Z_2 with at most one non-zero entry in each column. Let Φ' be defined by:

$$\Phi'_{i,j} = 1$$

i=candj=k$\Phi_{i,j} for some fixed$ cand k. $Then(\Phi') \geq (\Phi)$.

Proof: If $\Phi_{c,k} = 1$ then $\Phi' = \Phi$, and therefore $(\Phi') = (\Phi)$. Otherwise, if some other column $\Phi'_{*,x}$ has a non-zero in row c, then add $\Phi'_{*,x}$ to $\Phi'_{*,k}$ giving a new matrix Φ'', which has the same rank as Φ'. Since this other column must have at most one non-zero entry, the only change to Φ'' is that $\Phi''_{c,k} = 0$. Now $\Phi'' = \Phi$, and therefore $(\Phi'') = (\Phi)$. If no other column $\Phi_{*,x}$ has a non-zero in row c, then the rank of Φ' is one greater than that of Φ. We can use the preceding theorem to create a semi-perfect XOR-scheme with a decreased number of conflicts, by selectively adding ones to its matrix. In fact, the example given in Section V is a semi-perfect XOR-scheme. We call this process of selectively adding ones *augmenting*.

Given a perfect XOR-scheme which is not conflict free, we want to know if it can be augmented to a conflict free semi-perfect scheme. Unfortunately, answering this is NP-complete. Each column $\Phi_{*,i}$ will either be augmented or not. For each template T_j at most one column of $\Phi(T_j)$ may be augmented. Given these restrictions, we wish to find a subset of columns to augment such that all $\Phi(T_j)$ are augmented. For $p = 3$ this problem is exactly ONE-IN-THREE SAT, which is NP-complete.

VII. Heuristic Approaches

To p-color conflict graphs we extend the weight function to the edges and vertices of the graph. The weight of an edge is proportional to the number of extra CPU cycles that will be spent if the vertices of that edge are identically colored (assuming that all other edge constraints are met). Thus the weight of an edge is $\omega(v_i, v_j) = \sum_{v_i, v_j \in T_k} \omega(T_k)$, where $\omega(T_k)$ is the access frequency of T_k. The weight of a vertex is defined by $\omega(v_i) = \max_{v_j}\{\omega(v_i, v_j)\}$. The weighted graph for T is shown in Figure 2-c.

Graph coloring heuristics are used for solving problems such as scheduling, and register allocation. We present two heuristics called *Highest-Weighted-Conflict-First* (HWCF) and

Most-Immediate-Conflict-First (MISF) for generating perfect XOR-schemes. These are modifications of the basic greedy method for weighted graphs.

A color number is an integer in the range $[0 \dots p-1]$. A vertex number is an integer in the range $[0 \dots n-1]$. A template number is an integer in the range $[0 \dots t-1]$. An array $cost(v, c)$ indexed by vertex numbers and colors is used. Intially, $cost(v, c) = 0$ for all v and all c.

HWCF works as follows. Put all vertices in a priority queue H sorted in the decreasing order of the weight. Pick the maximally weighted vertex from H, say v, and assign it color c so that $cost(v, c)$ is the least among all colors (least conflict). The next step is to updates the cost values of the vertex's neighbors. For this $cost(w, c)$ is augmented by $\omega(w, v)$ for each vertex w that is adjacent to v and edge (w, v) is removed from the adjacency list as it will need no further consideration. Repeat until all vertices are expended.

We are best equipped to decide the color of a vertex when some of its neighbors have already been colored. The uncolored neighbors of vertices in the colored set are coloring candidates. MICF works as follows. Pick the maximally weighted vertex from H, say v, and color it as in HWCF. Next, add the vertices adjacent to v to a priority queue H_{adj} that is empty before any vertices are colored. We repeatedly remove the maximal vertex from H_{adj} and color it, until the current component is completely colored (the graph may not be connected). We have completely colored the current component. We proceed to the next component by deleting the colored vertices from H and setting $H_{adj} = \emptyset$. This proceeds until H becomes empty.

Analysis of the time complexity [5] shows that HWCF and MISF run in $O(p(t+n) + n^2 t)$ time, where t, 2^p, and n, are the number of templates, the number of processors, and the number of distinct vectors of the template bases, respectively.

We now introduce a heuristic SP for augmenting a perfect XOR-scheme, which has conflicts, to a semi-perfect scheme, with fewer conflicts. We assume that a perfect XOR-scheme is given. SP works as follows. If a one is added to a column $\Phi_{*,i}$, then a one cannot be added to any column $\Phi_{*,j}$, where $\Phi_{*,i}$ and $\Phi_{*,j}$ are both in some $\Phi(T_k)$. Or alternatively, v_i and v_j are both in T_k. Whenever a one is added to a column $\Phi_{*,i}$, mark all T_k such that v_i is in T_k as *blocked*.

Initially, no columns are blocked. We consider templates in order of their weight (maximal first). If T_i is conflict free, go to the next template. Otherwise, two columns of $\Phi(T_i)$ are necessarily identical. Adding a non-zero to one of the columns will increase the rank of $\Phi(T_i)$, provided that the row to which the non-zero is added contains only zeros, and that the column is not blocked. In the case that one of the columns is blocked, we augment the other one. If neither is blocked, we may choose to augment either. We choose the one which is contained in the least number of templates. Let the column so chosen be $\Phi_{*,j}$. We examine the rows of $\Phi(T_i)$. If some row k contains only zeros, we set $\Phi_{k,j} = 1$, and add all the vectors (columns) which appear in some template with v_j to the blocked set. We then proceed to the next template. The time complexity [5] of SP is $O(p^2 t \log t + \alpha p n)$ where α is the number of calls to SP.

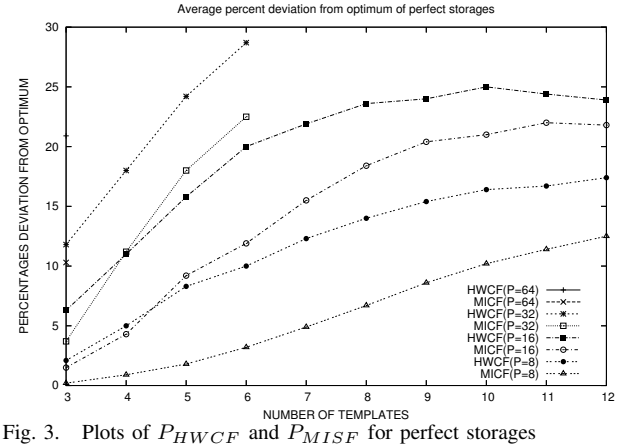

Fig. 3. Plots of P_{HWCF} and P_{MISF} for perfect storages

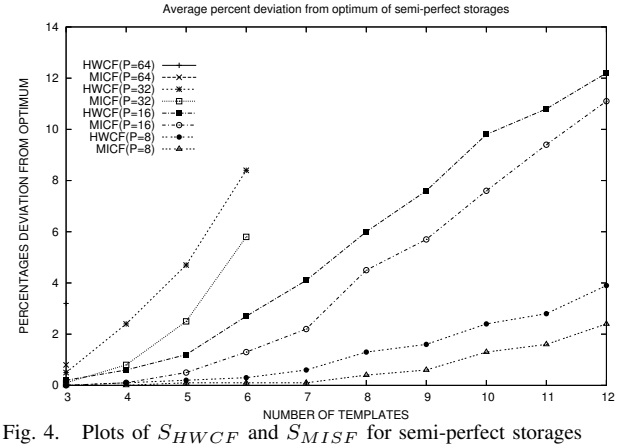

Fig. 4. Plots of S_{HWCF} and S_{MISF} for semi-perfect storages

VIII. Performance Evaluation

Let $\omega(T_i)$ be the number of accesses to T_i. A lower bound $A_{\min}$ on the number of accesses can be defined as $A_{\min} = \sum \omega(T_i)$. Unfortunately, there is no guarantee that $A_{\min}$ is achievable. Therefore, the optimum access time (A_{opt}) of a perfect scheme, for a given set of templates each with a given frequency, is found by using a branch-and-bound algorithm.

We evaluate the performance of a perfect XOR-scheme by comparing the number of accesses required with the optimal perfect XOR-scheme. Denote by A_s the average number of accesses of storage scheme s and let P_s be the percentage of extra memory accesses beyond the optimal that s requires. The number of memory accesses for a perfect XOR-scheme is:

$$A_s = \sum_{T_i \in T} \omega(T_i) 2^{(p - \text{rank}(\Phi(T_i)))}$$

The results of this simulation are displayed in Figure 3. Average P_{HWCF} and P_{MICF} are shown for $3 \le t \le 12$ and $8 \le P \le 64$. One thousand cases were run for each instance of P and t. The speed of the branch-and-bound algorithm prohibited us from completing the figures. Note that in general HWCF was outperformed by MICF. Both heuristics degrade in a smooth fashion with increasing numbers of templates and increasing template size. However, for a dozen templates and 16 memory modules, both heuristics deviate on the average

by more than 20% from the optimum solution.

The semi-perfect heuristic SP has been applied to each of the perfect schemes that are found using HWCF, MICF, and branch-and-bound, respectively. The percentage deviation from branch-and-bound of semi-perfect schemes are denoted by S_{HWCF} and S_{MICF}. Figure 4 shows the plots of S_{HWCF} and S_{MICF}. For example, using 6 templates and 32 memories the average increasing over the optimum in the access time (MICF) of a template is 0.058 cycles (Figure 4) if the optimum access is one cycle. The semi-perfect heuristic strongly reduces the degree of conflict and decreases its deviation by nearly 50% from that obtained for perfect-schemes.

Our scheme largely outperfoms row-major interleaving (6 to 18 times) and the rows, columns, and both diagonals of (4.23 to 5.84 times). It can easily be shown [5] that our approach to perfect storage finds optimum combined address transformations for arbitrary sets of power of 2 strides.

By considering a given set of templates, Frailong, Jalby, and Lenfant [3], analyzed the conditions for conflict-free access of data patterns. They proved that the columns or the rows of the needed address transformation matrix should be linearly independent. Our approach extends the concepts introduced by the above researchers and proposes an efficient method for *combining arbitrary templates* within one single address transformation. We have identified the necessary and sufficient conditions that a combined storage scheme should satisfy in order to minimize access conflicts.

Sohi [2] used manually synthesized address transformations (bit-wise) for stride access in vector processors. He proved that few memory buffers can reduce the effects of transient degradation in pipelined memories when arbitrary strides are accessed. By using bitwise linear transformation matrices, Boppana [4] proposed a conflict-free storage scheme to the row, column, main-diagonal, and square blocks. While all the above schemes are manually synthesized, we proposed heuristic approaches for automatic (compiler) synthesis of general storage schemes that minimize overall access time of arbitrary data templates. We also proposed heuristics for synthesizing storage schemes for arbitrary data templates.

IX. Conclusion

Our objective was to find heuristic approaches for automatic synthesis of general storage schemes. For this, we proved that finding the address transformation for a given set of data patterns is reduceable to graph coloring. We also proposed heuristics for synthesizing address transformations for arbitrary data patterns. Evaluation of this approach has experimentally proved to be effective in reducing the amount of conflicts while using reasonable implementation cost. The contribution of our work are: 1) a non-redundant XOR-matrix for arbitrary combined templates, 2) use of conflict graphs to represent the optimization problem, and 3) an efficient heuristic for minimizing the access time.

X. Acknowledgment

Thanks to the Research Committee and the College of Computer Science and Engineering, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for partially supporting this research.

References

[1] P. Budnik and D. Kuck. The organization and use of parallel memories. *IEEE Transactions on Computers*, C-20(12):1566–1569, Dec 1971.

[2] G. S. Sohi. High-bandwidth interleaved memories for vector processors–A simulation study. *IEEE Transactions on Computers*, 42(1):34–44, Jan 1993.

[3] J. M. Jalby W. Frailong and J. Lenfant. XOR-schemes: A flexible data organization in parallel memories. In *Proceedings of the International Conference on Parallel Processing*, pages 276–283, 1985.

[4] R. V. Boppana and C. S. Raghavendra. Efficient storage schemes for arbitrary size square matrices in parallel processors with shuffle-exchange networks. In *Proceedings of the International Conference on Parallel Processing*, pages 365–368, 1991.

[5] M. Al-Mouhamed and S. Seiden. A cost-effective heuristic storage for minimizing access time of arbitrary data templates. *Technical Report ICS-UCI 93-30, University of California, Irvine*, Jun 18 1993.